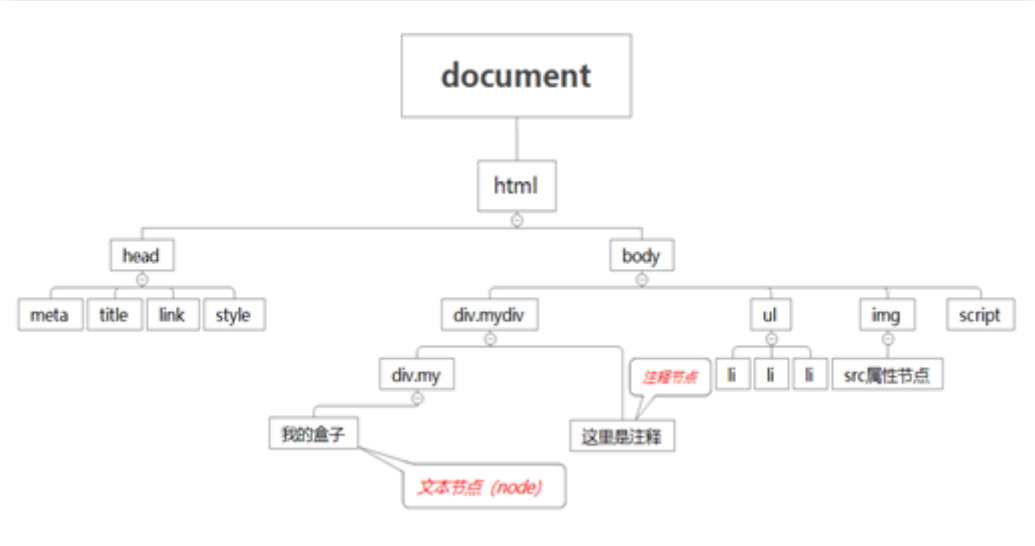
1、DOM概述：

全称Document Object Model文档对象模型

JS中通过DOM来对HTML文档进行操作，进行样式修改等

- 文档：整个HTML网页文档
- 对象：将网页中的每一个部分都转换为一个对象
- 模型：使用模型表示对象之间的关系，方便获取对象

DOM树：

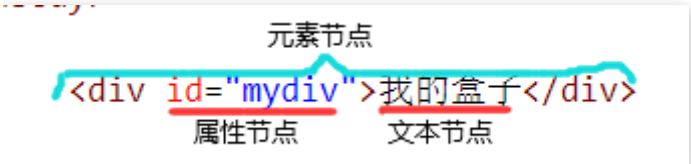


2、节点概述：

Node：是构成网页的基本组成部分，网页中的每一个部分都可以称之为节点。

EG：标签、属性、文本、注释等；但分类也不同，类型不同所使用的属性和方法都不尽相同

- 标签：元素节点
- 属性：属性节点
- 文本：文本节点
- 文档：文档节点



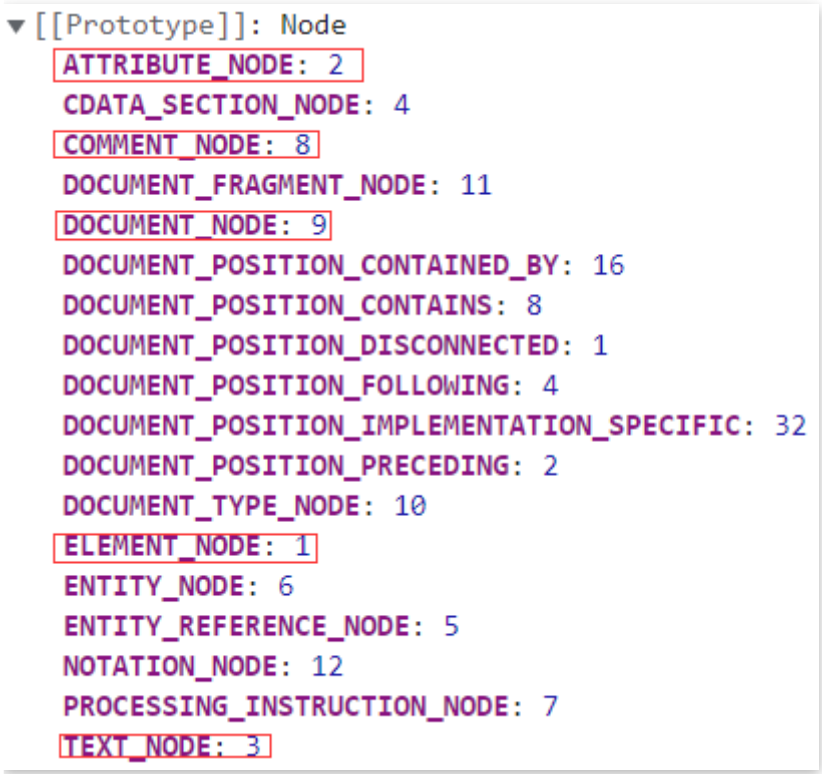
节点的关系：

- 父子
- 兄弟
- 祖先

节点属性：

	nodeName	nodeType	nodeValue
文档节点	#document	9	null
元素节点	标签名	1	null
属性节点	属性名	2	属性值
文本节点	#text	3	★文本内容

```
<div class="mydiv">2</div><!-- 注释节点 -->
<div id="mydiv">id</div>
<script>
    console.dir(document);
    //nodeName: "#document" 文档节点
    //nodeType: 9

    console.dir(document.getElementById('mydiv'));
    //nodeName: "DIV" 元素节点
    //nodeType: 1

    console.dir(document.getElementById('mydiv').childNodes);
    //nodeName: "#text" 文本节点
    //nodeType: 3

    console.dir(document.getElementsByClassName('mydiv')[0].nextSibling)
    //nodeName: "#comment" 注释节点
    //nodeType: 8

    console.log(document.querySelector('.mydiv').attributes)
    //nodeName: "class" 属性节点
    //nodeType: 2
```



3、文档节点（Document）

文档节点document，代表的是整个HTML文档，网页中的所有节点都是它的子节点

document对象作为window对象的属性存在的，我们不用获取可以直接使用

通过该对象我们可以在整个文档访问内查找节点对象，并可以通过该对象创建各种节点对象

1.document节点对象代表整个文档，每张网页都有自己的document对象。window.document属性就指向这个对象。

2.document对象的获取。

- 正常的网页，直接使用document或window.document。
- iframe框架里面的网页，使用iframe节点的contentDocument属性。
- Ajax 操作返回的文档，使用XMLHttpRequest对象的responseXML属性。
- 内部节点的ownerDocument属性。

3.对于 HTML 文档来说，document对象一般有两个子节点。

- 第一个子节点是document.doctype，指向<DOCTYPE>节点，即文档类型（Document Type Declaration，简写DTD）节点。document.firstChild通常就返回这个节点。
- 第二个子节点document.documentElement属性返回当前文档的根元素节点（root）。

4.document.body属性指向<body>节点。

5.document.head属性指向<head>节点。

6.document.forms属性返回所有<form>表单节点。

7.document.images属性返回页面所有<img>图片节点。

8.document.embeds属性和document.plugins属性，都返回所有<embed>节点

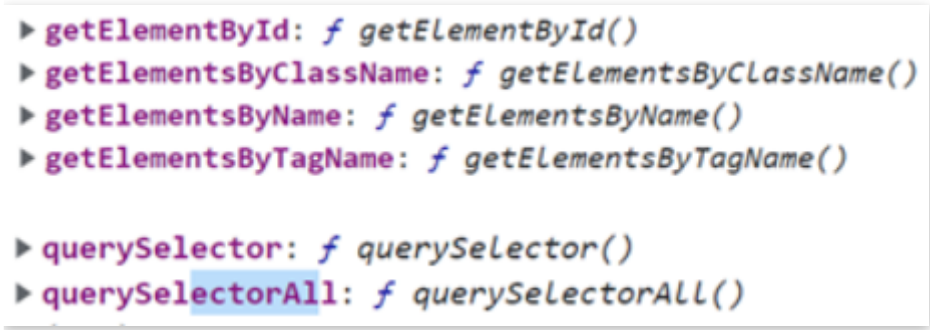
9.document.scripts属性返回所有<script>节点。

10.document.styleSheets属性返回文档内嵌或引入的样式表集合(CSS)。

11.document.documentURI属性返回一个字符串，表示当前文档的网址。documentURI继承自Document接口，可用于所有文档。
12.document.URL属性都返回一个字符串，表示当前文档的网址。URL继承自HTMLDocument接口，只能用于 HTML 文档。
更多属性方法见：(2条消息) 文档document节点，属性与方法_zdw火车叨位去的博客-CSDN博客

4、元素节点（Element）

HTML中的各种标签都是元素节点，这也是我们最常用的一个节点；浏览器会将页面中所有的标签都转换为一个元素节点，我们可以通过document的方法来获取元素节点
比如：



注意：DOM方法中除了getElementById和querySelector方法，其他方法获取的都是类数组
获取页面节点的方法：

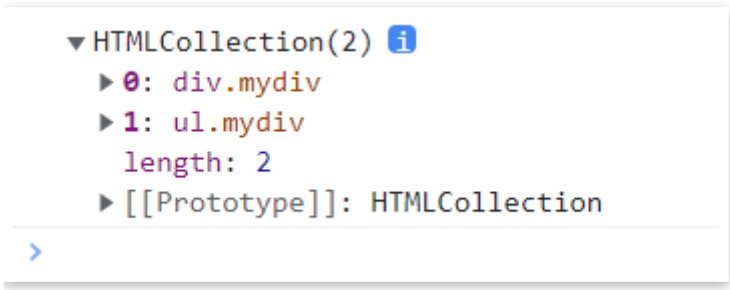
1、使用内置的方法来获取：

- document.getElementById()：根据id属性值获取一个元素节点对象

```
<div id="mydiv">我的盒子</div>
    <script>
        let mydiv = document.getElementById('mydiv');
        console.dir(mydiv);
    </script>
```

- document.getElementsByClassName()：根据class属性值获取一个元素节点对象

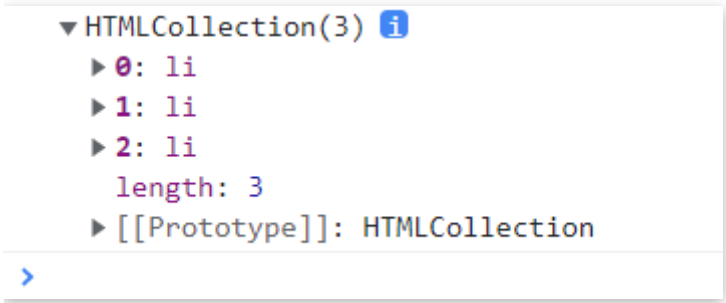
```
<div class="mydiv">我的盒子</div>
    <span class="my">poppoppop</span>
    <ul class="mydiv">
        <li>1</li>
        <li>2</li>
        <li>3</li>
    </ul>
    <script>
        let mydiv1 = document.getElementsByClassName('mydiv');
        console.dir(mydiv1);
    </script>
```



- document.getElementsByTagName：根据标签获取节点，返回的是一个类数组

```
<div class="mydiv">我的盒子</div>
    <span class="my">poppoppop</span>
    <ul class="mydiv">
```

```
        <li>1</li>
        <li>2</li>
        <li>3</li>
    </ul>
    <script>
        let lis = document.getElementsByTagName('li');
        console.dir(lis);
    </script>
```



- document.getElementsByName：根据标签获取节点，返回的是一个类数组

```
<input type="text" name="username">
<input type="checkbox" name="hobby"> 学习
<input type="checkbox" name="hobby"> 运动
<input type="checkbox" name="hobby"> 打游戏
<script>
    //返回类数组，即便只有一个元素
    let input = document.getElementsByName('username');
    console.log(input); //NodeList [input]


    //返回类数组
    let checkbox = document.getElementsByName('hobby');
    console.log(checkbox); //NodeList(3) [input, input, input]
</script>
```

- document.querySelector()：支持各种选择器，返回第一个满足条件的节点；
- document.querySelectorAll()：返回所有的满足条件的节点

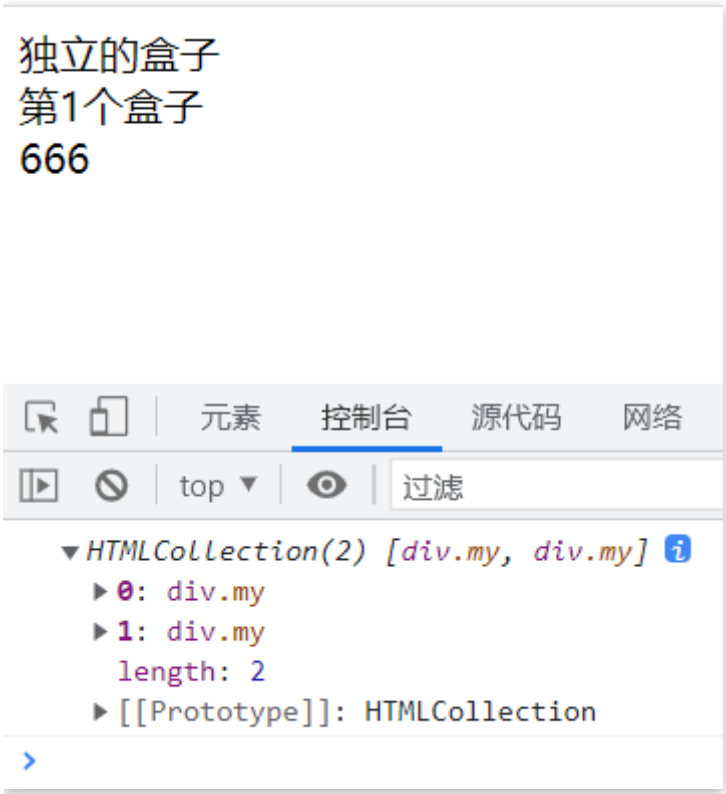
```
<input type="text" name="username">

<input type="checkbox" name="hobby"> 学习
```

```
<input type="checkbox" name="hobby"> 运动
<input type="checkbox" name="hobby"> 打游戏

<script>
    //返回的是节点
    let input = document.querySelector('input[name="username"]');
    console.log(input); //<input type="text" name="username">

    //Notice
    let hobby = document.querySelector('input[name="hobby"]');
    console.log(hobby); //<input type="checkbox" name="hobby">

    //获取到所有的满足条件的节点
    let hobbyArr = document.querySelectorAll('input[name="hobby"]');
    console.log(hobbyArr); //0: input 1: input 2: input
</script>
```

2、 根据节点的关系来获取：

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>document</title>
    <style>
        #mydiv {
            width: 300px;
            height: 50px;
            background-color: #f01;
            margin: 50px;
        }
    </style>
</head>

<body>
    <div class="my">
        独立的盒子
    </div>

```

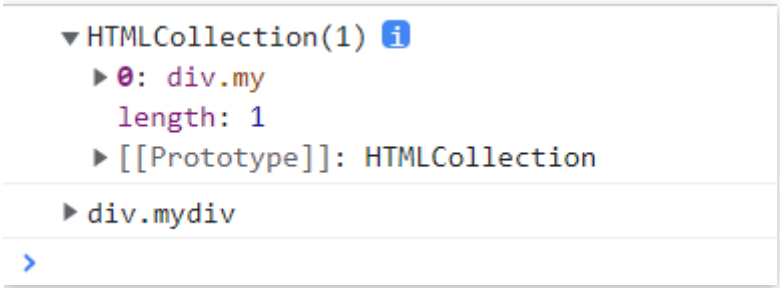
```
    <div class="mydiv">
        <div class="my">
            第1个盒子
        </div>
        <div class="my">
            第2个盒子
        </div>
    </div>

    <script>
        //注意下标
        let mydiv = document.getElementsByClassName('mydiv')[0];
        console.log(mydiv.getElementsByClassName('my'));
        //HTMLCollection(2) [div.my, div.my]
        mydiv.getElementsByClassName('my')[1].innerHTML = '666';
    </script>
</body>
```



- 父节点：parentNode

```
<div class="mydiv">
     <div class="my">
        第1个盒子
     </div>
```

```
</div>

<script>
     let my1 = document.getElementsByClassName('my');
     console.dir(my1);
     console.dir(my1[0].parentNode); //获取父节点
        //div.mydiv
</script>
```



- 获取所有父级节点：

```
<div class="mydiv">
    <div class="my">
        <div class="span">
            <div class="lll">
                100
            </div>
        </div>
    </div>
</div>

<script>
    let lll = document.getElementsByClassName('lll')[0];
    console.dir(lll);

    Element.prototype.allParents = function () {
    let arr = [];
    let pNode = this.parentNode;
    while (pNode && pNode != document) {
         arr.push(pNode);
         pNode = pNode.parentNode;
         }
         arr.pop();
         arr.pop();
```

```
        return arr;
    }
    console.log(lll.allParents());

    console.log(document.getElementsByClassName('my')[0].allParents());
    //['div.span', 'div.my', 'div.mydiv', 'body', 'html']
</script>
```

- 子：children child

```
<div class="mydiv">
    一个盒子
    <div class="my">
        儿子
    </div>
    <!-- 注释节点 -->
    <span>
        文本信息
    </span>
</div>
<script>
    let mydiv = document.getElementsByClassName('mydiv');
    console.log(mydiv[0].childNodes);
```

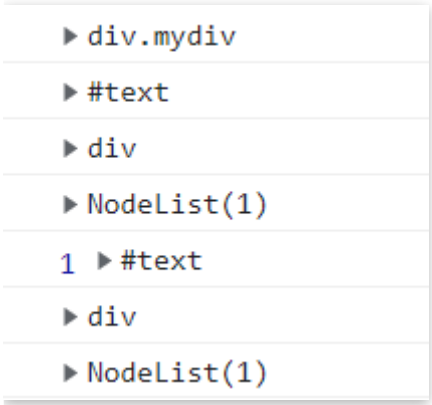
```
    //是个属性，获取的是子节点

    console.log(mydiv[0].children);
    //是个属性，获取的是子元素
</script>
```

- 查找 HTML 父子：

方法	描述
元素节点.parentNode	返回元素的父节点。
元素节点.parentElement	返回元素的父元素。
元素节点.childNodes	返回元素的一个子节点的数组（包含空白文本Text节点）。
元素节点.children	返回元素的一个子元素的集合（不包含空白文本Text节点）。
元素节点.firstChild	返回元素的第一个子节点（包含空白文本Text节点）。
元素节点.firstElementChild	返回元素的第一个子元素（不包含空白文本Text节点）。
元素节点.lastChild	返回元素的最后一个子节点（包含空白文本Text节点）。
元素节点.lastElementChild	返回元素的最后一个子元素（不包含空白文本Text节点）。
元素节点.previousSibling	返回某个元素紧接之前节点（包含空白文本Text节点）。
元素节点.previousElementSibling	返回指定元素的前一个兄弟元素（相同节点树层中的前一个元素节点）。
元素节点.nextSibling	返回某个元素紧接之后节点（包含空白文本Text节点）。
元素节点.nextElementSibling	返回指定元素的后一个兄弟元素（相同节点树层中的下一个元素节点）。

```
<div>3</div>
    <div class="mydiv">1</div>
    <div>2</div>
    <script>
        let mydiv = document.getElementsByClassName('mydiv')[0];
        console.dir(mydiv);

        //下一个兄弟节点
        console.log(mydiv.nextSibling);

        //下一个兄弟元素
        console.dir(mydiv.nextElementSibling);

        //下一个兄弟元素 de 子节点
        console.dir(mydiv.nextElementSibling.childNodes);

        //上一个兄弟节点
        console.log(1, mydiv.previousSibling);

        //上一个兄弟元素
```

```
        console.dir(mydiv.previousElementSibling);

        //上一个兄弟元素 de 子节点
        console.dir(mydiv.previousElementSibling.childNodes);
    </script>
```



- 节点关系：sibling.all

```
    <div>3</div>
    <div class="mydiv">2</div>
    <div>1</div>
    <ul>
        <li></li>
        <li></li>
        <li></li>
    </ul>
    <a href="">连接</a>
    <script>
        let mydiv = document.getElementsByClassName('mydiv')[0];
        Element.prototype.allSiblings = function () {
            return [...this.parentNode.children];
        }
        console.log(mydiv.allSiblings()); //拿到所有的兄弟元素
        //[div, div.mydiv, div, ul, a, script]

        //用fitler()过滤
        Element.prototype.allSiblings = function () {
            return [...this.parentNode.children].filter(e => e != this);
            //this指向的是距离他最近的function的调用者
        }
        console.log(mydiv.allSiblings()); //拿到所有的兄弟元素
```

```
        //[div, div, ul, a, script]

        //直接写成一个函数
        function allElementSiblings(node) {
            //this指向的是距离他最近的function的调用者
            return [...node.parentNode.children].filter(e => e != node);
        }
        console.log(allElementSiblings(mydiv));
        //[div, div, ul, a, script]
        </script>
```

- 更多节点获取方法：

```
<div>
        <div>3</div>
        <div class="mydiv">1</div>
        <div>2</div>
        <ul>
            <li></li>
            <li></li>
            <li></li>
        </ul>
        <a href="">连接</a>
        <img src="./1.png" alt="">
        <img src="./2.png" alt="">
        <img src="./3.png" alt="">

        <form action=""></form>
        <form action=""></form>
        <form action=""></form>
    </div>


    <div id="mydiv">
        id
    </div>
    <script>
        //快捷方式-->就是属性的熟悉，不算是知识点学习
        console.dir(document);
```

```
        console.log(document.body);
        console.log(document.images);
        console.log(document.forms);
        console.log(document.getElementById('mydiv'));
        //console.log(document.mydiv); //undefined
    </script>
```

```
▶ #document
▶ <body>…</body>
▶ <script>…</script>
▶ HTMLCollection(3) [img, img, img]
▶ HTMLCollection(2) [form, form]
  <div id="mydiv">id</div>
```

5、文本节点（Text）：

文本节点表示的是HTML标签以外的文本内容，任意非HTML的文本都是文本节点

它包括可以字面解释的纯文本内容

文本节点一般是作为元素节点的子节点存在的

获取文本节点时，一般先要获取元素节点，再通过元素节点获取文本节点。例如：元素节点.firstChild;

获取元素节点的第一个子节点，一般为文本节点

1.nodeType的值为3。

2.nodeName的值为“#text”。

3.nodeValue或者的值为节点所包含的文本。

4.parentNode是一个Element。

5.appendData(text):将text添加到节点的末尾。

6.deleteData(offset，count)：从offset指定的位置开始删除count个字符。

7.insertData(offset,text):在offset指定的位置插入text。

8.replaceData(offset,count,text):用text替换从offset指定的位置开始到offset+count为止出的文本。

9.splitText(offset):从offset指定的位置将当前文本节点分成俩个文本节点。

10.substringData(offset,count):提取从offset指定的位置开始到offse+count位置处的字符串。

11.createTextNode():创建文本节点。

12.normalize()合并文本节点。

13.splitText()分割文本节点。

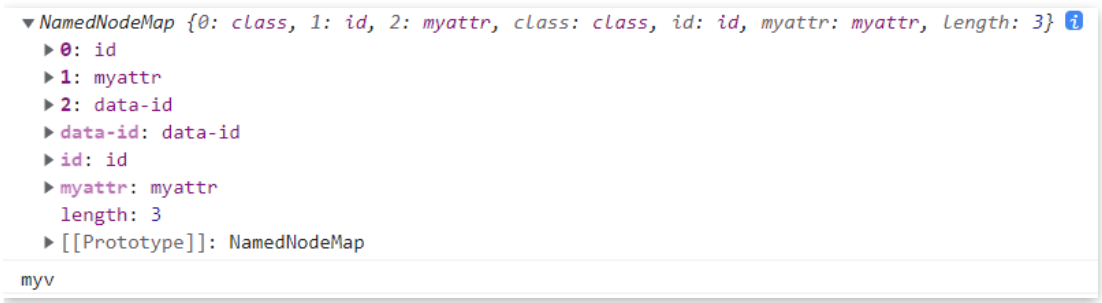
原文链接：https://blog.csdn.net/weixin_28686771/article/details/111977451

6、属性节点（Attr）：

属性节点表示的是标签中的一个一个的属性，这里要注意的是属性节点并非是元素节点的子节点，而是元素节点的一部分，可以通过元素节点来获取指定的属性节点。例如：元素节点.getAttributeNode("属性名")；浏览器已经为我们提供文档节点对象，这个对象是window；属性可以在页面中直接使用，文档节点代表的是整个网页。

```
setAttribute(‘属性名’, ‘属性值’)  // 设置指定的属性名的值
getAttribute()  // 获取指定的属性名的值 attributes属性获取所有
hasAttribute()  // 检查一个属性是否存在
removeAttribute()  // 删除指定的属性


<div data-id="6" name="'111" id="sss" class="ccc"></div>
var div = document.getElementsByTagName('div')[0];
// 1. 获取属性
console.log(div.attributes, div.attributes[0].value); // 获取所有属性 某个属性
console.log(div.getAttribute(name));
console.log(div.getAttribute('name'));

// 2. 设置属性(新增/修改)
div.attributes[0].value = '9'; // 修改属性成功
div.setAttribute('name', 'florence');  // 修改
div.setAttribute('data-arr', '333'); // 新增

// 3. 检测是否包含某个属性
console.log(div.hasAttribute('name')); // true

// 4. 删除某个属性
div.removeAttribute('name');
console.log(div.hasAttribute('name')); // false

<div class="mydiv" id="myv">22222</div>
    <script>
        let my = document.getElementById('myv');
        console.log(my.attributes);

        //万能的属性的操作:增(set)删(remove)改(set)查(get)
        //用起来相对来讲比较复杂
        my.setAttribute('data-id', 300); //增
```

```
        my.setAttribute('mydiv', 'newvalue'); //改
        my.removeAttribute('class'); //删除
        console.log(my.getAttribute('id')); //查
    </script>
```



标准属性：

```
/*
1. A：href、target；
2. DIV：id;
3. img：src;
4. input的value;
5. checkbox的checked
6. ...
7. 标准属性可以直接使用：节点.属性名; 如:节点.src = ‘./topimg1.jpg’;
**注意：class 属性不能直接操作**
*/
<div class="mydiv" id="myv" myattr="myv">22222</div>
<img src="../img/1.jpg" alt="" style="width:100px;height:100px">

<input type="text" value="300" name="username">
<script>
 let my = document.getElementById('myv');
     console.log(my.id);
     console.log(document.querySelector('img').src);
     document.querySelector('img').src
='https://gimg2.baidu.com/image_search/src=http%3A%2F%2Fimg.jj20.com%2Fup%2Fallimg%2F1114%2F063021120F9%2F210630120F9-1-1200.jpg&refer=http%3A%2F%2Fimg.jj20.com&app=2002&size=f9999,10000&q=a80&n=0&g=0n&fmt=auto?sec=1656142066&t=a3a8b36573cd1666beb7df4571715a5d';

     console.dir(document.querySelector('input[name="username"]'));
     document.querySelector('input[name="username"]').value = '蓝鲸';
</script>
```

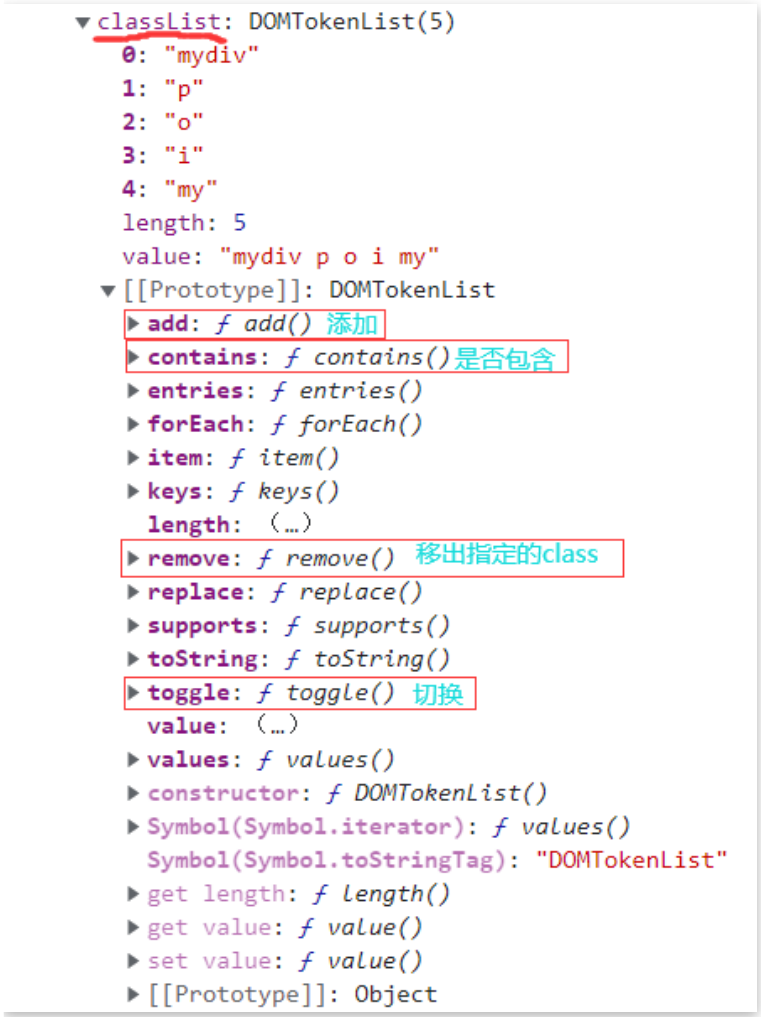
class属性:

```
<div class="mydiv p o i my">22222</div>
<script>
    let my = document.getElementsByClassName('mydiv')[0];
    console.log(my.className); //mydiv p o i my

    //classList
    console.dir(my); //div.mydiv.p.o.i.my
    //增
    my.classList.add('b');
    console.log(my.className); //mydiv p o i my b
    //删
    my.classList.remove('i');
    console.log(my.className); //mydiv p o my b

    //是否包含
    console.log(my.classList.contains('p')); //true
    console.log(my.classList.contains('my')); //true
    console.log(my.classList.contains('my1')); //false

    //toggle:有的删除，没有的追加
    my.classList.toggle('mydiv');
    console.log(my.className); //p o my b
    my.classList.toggle('gg');
    console.log(my.className); //p o my b gg
</script>
```

```
// 一次性修改元素的classList
el.className;

// classList：节点的classList提供了操作class的更专业的方法：
classList.add()  // 在不影响原始class的情况下追加新的class；
classList.remove()  // 删除指定的class，没指定的不会删除；
classList.contains()  // 检查是否存在指定的class；
classList.replace() // 将前者类名替换为后者类名

```

```
<div class="name1 name2"></div>
<script>


    var div = document.getElementsByTagName('div')[0];
    console.log(div.className); // 1. 获取
    console.log(div.classList); // 1. 获取

    div.className = 'www mmm sss'; // 2. 修改
    div.classList.replace('rrr', 'uuu'); // 2. 修改 将div的className为rrr的替换为uuu
    div.classList[0] = 'sdf'; // 2. 修改不成功

    div.className = 'www mmm'; // 3. 删除
    div.classList.remove('www'); // 3. 删除

    div.className = 'www mmm ddd'; // 4. 新增
    div.classList.add('rrr'); // 4. 新增

    console.log(div.classList.contains('rrr')); // 5. 检测 true
    console.log(div.classList.contains('www')); // false
</script>
```

classList:



自定义属性:

```
// 获取自定义属性的值，例如标签的data-myattr = ‘value’
el.dataset.myattr;

<div data-id="6"></div>
var div = document.getElementsByTagName('div')[0];
console.log(div.dataset.id); // 1. 获取
div.dataset.id = 9; // 2. 更改
div.dataset.name = 'Florence'; // 3. 新增
delete div.dataset.name; // 4. 删除

 <div class="mydiv i my" data-uid="20" data-id="30">4545445</div>
 <script>
      let my = document.getElementsByClassName('mydiv')[0];
      console.dir(my); //div.mydiv.i.my
      console.dir(my.dataset.uid); //20
      console.dir(my.dataset.id); //30
```

```
</script>
```

自定义属性：使用场景

```
<div>
        <p>文章主题
            <br>
            文章标题
            <br>
            <div class="support" data-aid="31">点赞</div>
            <div class="collect" data-aid="31">收藏</div>
        </p>
        <p>文章主题
            <br>
            文章标题
            <br>
            <div class="support" data-aid="31">点赞</div>
            <div class="collect" data-aid="31">收藏</div>
        </p>
        <p>文章主题
            <br>
            文章标题
            <br>
            <div class="support" data-aid="31">点赞</div>
            <div class="collect" data-aid="31">收藏</div>
        </p>
    </div>
    <script>
        let support = document.querySelectorAll('.support');
        support.forEach(function (foo) {
            foo.onclick = function () {
                console.log(this.dataset.aid);
            }
        });
    </script>
```

7、获取、修改HTML的值：

1. **获取HTML的值：**

方法	描述
元素节点.innerHTML	获取 HTML 元素的 inner HTML
元素节点.innerText	获取 HTML 元素的 inner Text
元素节点.属性	获取 HTML 元素的属性值。
元素节点.getAttribute(attribute)	获取 HTML 元素的属性值。
元素节点.style.样式	获取 HTML 元素的行内样式值。
元素节点.src / href	获取 HTML 元素的连接地址

在其它浏览器中可以使用getComputedStyle()这个方法来获取元素当前的样式，这个方法是window的方法，可以直接使用，但是需要两个参数：

第一个参数：要获取样式的元素

第二个参数：可以传递一个伪元素，一般都传null

```
<head>
    <meta charset="UTF-8">
    <title></title>
    <style>
        /*样式表的样式*/
        #box {
            width: 200px;
            height: 200px;
            background-color: green;
        }
    </style>
</head>
<body>
<div style="width: 100px;height: 100px;" id="box"></div>

<script>
    /*通用的获取元素样式的方法*/
    function getStyle(obj, name) {
        if (window.getComputedStyle) {
            //正常浏览器的方式，具有getComputedStyle()方法
            return getComputedStyle(obj, null)[name];
        } else {
```

```
            //IE8的方式，没有getComputedStyle()方法
            return obj.currentStyle[name];
        }
    }

    var box = document.getElementById("box");

    console.log(getStyle(box, "width"));
    console.log(getStyle(box, "height"));
    console.log(getStyle(box, "background-color"));
</script>
</body>
```

编写一段兼容性代码，用来获取任意标签的文本内容：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
<a href="https://www.baidu.com" id="a">打开百度，你就知道！</a>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var a = document.getElementById("a");

    console.log(getInnerText(a));

    /*获取任意标签的内容*/
    function getInnerText(element) {
        // 判断浏览器是否支持textContent,如果支持，则使用textContent获取内容，否则使用innerText获取内容。
        if(typeof element.textContent == "undefined") {
            return element.innerText;
        } else {
            return element.textContent;
        }
```

```
    }
</script>
</body>
</html>
```

2. 改变 HTML 的值:

方法	描述
元素节点.innerText = *new text content*	改变元素的 inner Text。
元素节点.innerHTML = *new html content*	改变元素的 inner HTML。
元素节点.属性 = *new value*	改变 HTML 元素的属性值。
元素节点.setAttribute(*attribute*, *value*)	改变 HTML 元素的属性值。
元素节点.style.样式 = *new style*	改变 HTML 元素的行内样式值。

1 innerText

```
<button id="btn">我是按钮</button>

<!-- 在这里写JavaScript代码，因为JavaScript是由上到下执行的 -->
<script>
    var btn = document.getElementById("btn");
    btn.innerText = "我是JavaScript的按钮";
    console.log(btn);
</script>

```

2 innerHTML

```
<div id="box"></div>
<script>
    var box = document.getElementById("box");
    box.innerHTML = "<h1>我是Box中的大标题</h1>";
    console.log(box);

    document.body.innerHTML += '<div>111</div>';
    // 可以在页面中一一加入创建的元素，不会覆盖上一次添加的元素
</script>

```

```
// 包括内部html标签，修改符为= 和 +=  可对内容进行新增、修改、删除
el.innerHTML;
document.body.innerHTML += '<div>111</div>'; // 新增
document.body.innerHTML = ''; // 清空内容
document.body.innerHTML = '<li></li>'; // 修改/替换

// 包括内部html标签，还包括自己
el.outerHTML

// 返回文本，没有html标签，赋值要小心
el.innerText

```

3

```
<a id="a" href="">打开百度，你就知道！</a>

<script>
    //1
    var a = document.getElementById("a");
    a.href="https://www.baidu.com";
    console.log(a);

    //2
    var a = document.getElementById("a");
    a.setAttribute("href", "https://www.baidu.com");
    console.log(a);
</script>
```

3. 其他方法

修改节点的内容除了常用的innerHTML和innerText之外，还有insertAdjacentHTML和insertAdjacentText方法，可以在指定的地方插入内容。insertAdjacentText方法与insertAdjacentHTML方法类似，只不过是插入纯文本，参数相同。

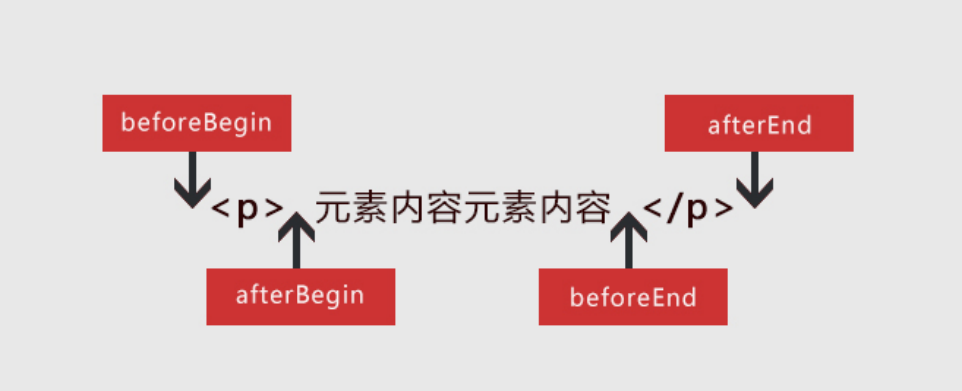
语法说明：

```
object.insertAdjacentHTML(where,html);
object.insertAdjacentText(where,text)
```

参数说明：

1. where：

- beforeBegin：插入到开始标签的前面
- beforeEnd：插入到结束标签的前面
- afterBegin：插入到开始标签的后面
- afterEnd：插入到结束标签的后面



2. html：一段html代码
3. text：一段文本值

注意事项：

- 这两个方法必须等文档加载好后才能执行，否则会出错。
- insertAdjacentText只能插入普通文本，insertAdjacentHTML插入html代码。
- 使用insertAdjacentHTML方法插入script脚本文件时，必须在script元素上定义defer属性。
- 使用insertAdjacentHTML方法插入html代码后，页面上的元素集合将发生变化。
- insertAdjacentHTML方法不适用于单个的空的元素标签(如img，input等)。

案例演示：

```
<div id="insert">
    <p>为书法的人</p>
</div>

<script>
    var div = document.getElementById("insert");
    div.insertAdjacentHTML('beforeBegin', 'IIS你家：');
</script>
```

编写一段兼容性代码，用来设置任意标签的文本内容

```
<a href="https://www.baidu.com" id="a">打开百度，你就知道！</a>

<script>
   var a = document.getElementById("a");
```

```
    setInnerText(a, "你要打开百度吗？");
        console.log(getInnerText(a));

        /*获取任意标签的内容*/
        function getInnerText(element) {
            // 判断浏览器是否支持textContent,如果支持，则使用textContent获取内容，否则使用innerText获取内容。
            if (typeof element.textContent == "undefined") {
                return element.innerText;
            } else {
                return element.textContent;
            }
        }

        /*设置任意标签的内容*/
        function setInnerText(element, text) {
            // 判断浏览器是否支持textContent,如果支持，则使用textContent设置内容，否则使用innerText设置内容。
            if (typeof element.textContent == "undefined") {
                return element.innerText = text;
            } else {
                return element.textContent = text;
            }
        }
    </script>
```